

Date:

Ex :No.1

STUDY OF SIMULATION TOOLS

AIM : To study about simulation tools using Verilog language for logic gates and verify the output.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

THEORY:

Modelism simulator:

Simulation tests a model of the system we wish to build. Simulation checks two properties

- i. Functional correctness
- ii. Timing correctness

Hardware Description Language (HDL) can be used to modal a digital system.

Commonly used HDLs are

- i. VHDL - Very High speed Integrated Circuit Hardware Description Language.
- ii. Verilog HDL.

Verilog HDL is a Hardware Description Language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level to the switch level.

The Verilog HDL Language Includes capabilities to describe the behavioral nature of a design, the dataflow nature of a design, a design's structural composition, delays and a waveform generation mechanism including aspects of response monitoring and verification, all modeled using one single language.

The basic unit of description in Verilog is the module. The module describes the functionality or structure of a design It also describes the ports through which if communicates externally with other modules.

Xilinx Integrated Software Environment (ISE) is a powerful yet flexible integrated design environment that allows us to design Xilinx FPGA devices from start to finish ISE includes our world class design entry, synthesis and implementation tools delivering the industry's fastest place and route times, highest performance, and most advanced design methodologies.

Project Navigator is the user interface that helps us manage the entire

design process including design process including design entry, simulation, synthesis and implementation.

AND Gate:

And gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. $A.B$.

Truth table:

2 Input AND gate		
A	B	$Y=A.B$
0	0	0
0	1	0
1	0	0
1	1	1

Symbol:



OR gate:

OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

Truth table:

2 Input OR gate		
A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1
1	1	1

Symbol:



NOT gate:

NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A the inverted output is known as NOT A. This is also shown as A' or A with a bar over the top, as shown at the outputs.

Truth Table:

NOT gate	
A	$Y = \overline{A}$
0	1
1	0

Symbol:



NAND Gate:

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Truth table:

2 Input NAND gate		
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Symbol:



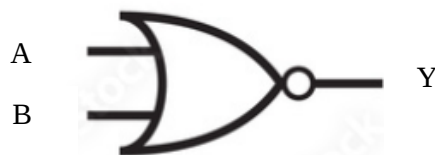
NOR gate:

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an Or gate with a small circle on the output. The small circle represents inversion.

Truth table:

2 Input NOR gate		
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Symbol:



EXOR gate

The 'Exclusive -OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign ($\oplus$) is used to show the EXOR operation.

Truth table

2 Input EXOR gate		
A	B	$Y=A\oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Symbol:



EXNOR gate

The 'Exclusive-NOR' gate circuit does the opposite to the EXOR gate. It will give a low output if either, but not both of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

Truth table:

2 Input EXNOR gate		
A	B	$Y=A\odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Symbol:



PROCEDURE

1. Click on Start -> All Programs -> Xilinx Designer Tools -> Project Navigator.
2. Click on File -> New Project.
3. Select a Name and Location, then click -> Next.
4. In the Project Setting use the following configuration.

Family as Spartan 3E
Device as XC3S100E
Package as TQ144
Speed as -4
5. Click on -> Next -> Finish.
6. Click on Project -> New Source -> Verilog Module and type a filename.
7. Click on -> Next -> Finish.
8. Type your source file and save.
9. Click on Simulation radio button on the left corner of the screen.
10. Select your filename in the hierarchy.
11. Expand ISIM Simulator and check for behaviour and simulate the program.
12. Select the necessary input values by right click on value -> Force Constant .
13. Enter the value in the Force to Value dialog box .
14. Click on "Run for specific time" in the toolbar.
15. Observe the waveforms.

Program :

AND GATE:

```
module and_gate (a,b,y);  
input a,b;  
output y;  
assign y = a & b;  
endmodule
```

OR GATE:

```
module or_gate (a,b,y);  
input a,b;  
output y;  
assign y = a | b;  
endmodule
```

XOR GATE:

```
module xor_gate (a,b,y);  
input a,b;  
output y;  
assign y = a ^ b;  
endmodule
```

NOT GATE:

```
module not_gate (a,b) ;  
input a;  
output b;  
assign b = ~ a;  
endmodule
```

NAND GATE:

```
module nand_gate (a,b,y);  
input a,b;  
output y;  
assign y = ~ (a & b);  
endmodule
```

NOR GATE:

```
module nor_gate (a,b,y);  
input a,b;  
output y;  
assign y = ~ (a | b);  
endmodule
```

EXNOR GATE:

```
module exnor_gate (a,b,y);  
input a,b;  
output y;  
assign y = ~ (a ^ b);  
endmodule
```

Result:

Date:

Ex No : 2 SIMULATION AND IMPLEMENTATION OF 8-BIT ADDER

AIM : To simulate and implement 8-bit Ripple Carry adder in FPGA using Verilog code.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

THEORY:

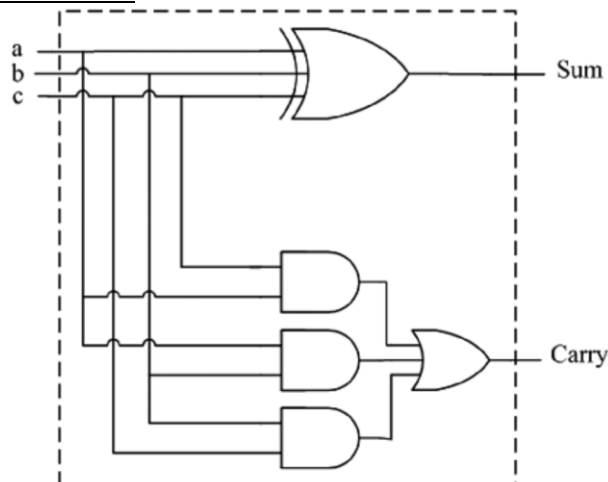
FULL ADDER:

Full adder is a logic circuit that performs addition operation on three bit binary numbers. Full adder produces the sum of three inputs and carry value.

TRUTH TABLE :

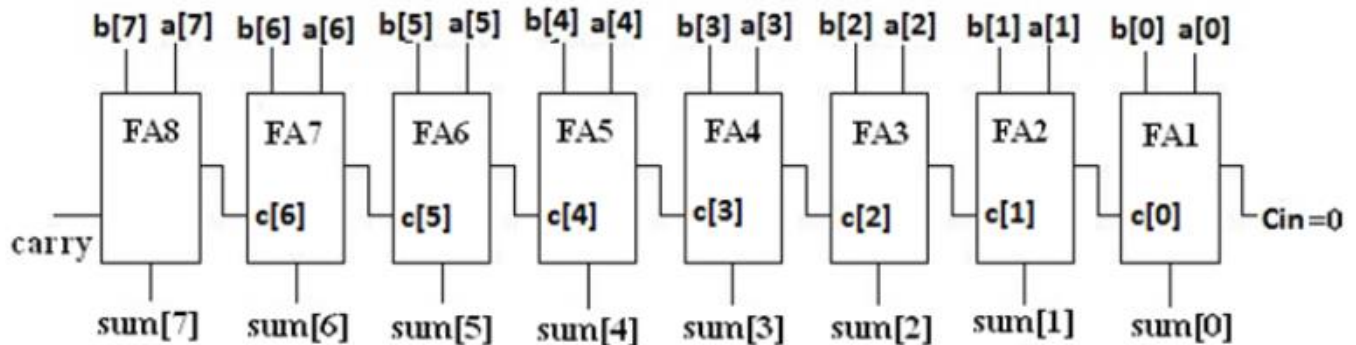
a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

LOGIC DIAGRAM:



8-BIT RIPPLE CARRY ADDER USING FULL ADDER

A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry-in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. A standard 8-bit ripple-carry adder built as a cascade from eight 1-bit full-adders.



PROCEDURE:

SIMULATION :

1. Click on Start -> All Programs -> Xilinx Designer Tools -> Project Navigator.
2. Click on File -> New Project.
3. Select a Name and Location, then click -> Next.
4. In the Project Setting use the following configuration.

Family as Spartan 3E
Device as XC3S100E
Package as TQ144
Speed as -4

5. Click on -> Next -> Finish.
6. Click on Project -> New Source -> Verilog Module and type a filename.
7. Click on -> Next -> Finish.

8. Type your source file and save.
9. Click on Simulation radio button on the left corner of the screen.
10. Select your filename in the hierarchy.
11. Expand ISIM Simulator and check for behaviour and simulate the program.
12. Select the necessary input values by right click on value -> Force Constant.
13. Enter the value in the Force to Value dialog box.
14. Click on "Run for specific time" in the toolbar.
15. Observe the waveforms.

IMPLEMENTATION:

- Click on project, go to new source. Select implementation constraints file type and enter file name & click next, then finish
- Write the inputs, outputs & its pin location in proper format of .ucf file.
- Open main program & double click on synthesize-XST. After successful completion of synthesis, double click on implementation design. (Translate, Map, Place & Route)
- After Successful completion of implement design double click on Programming file Generation.
- After the successful completion of all these processes double click on configure Target Device and a new ISE iMPACT window open.
- Connect the Xilinx board to your PC using USB board.
- Double click on Boundary Scan. Check auto cable connection out.
- Click on File->Initialize chain.
- After that they ask for "Do You want to continue and assign configuration files?"
- Click on Yes and Select the generated bit-stream file.
- Click on open. After that they ask "Do you want to attach an SPI or BPI PROM to this device". Click on operation -> program. If programming successful then they show Program succeeded.
- Output can be viewed in the board by using LEDs and switches.

```
//FULL ADDER
```

```
module fulladder (a,b,c,sum,cout);
```

```
output sum, cout;
```

```
input a,b,c;
```

```
assign sum = a^b^c;
```

```
assign cout=(a&b)|(b&c)|(c&a);
```

```
endmodule
```

```
//8-BIT RCA
```

```
module ripple_8adder( input [7:0] a, input [7:0] b, input cin, output [7:0] sum,  
output carry );
```

```
wire [6:0]c;
```

```
fulladder FA1(a[0],b[0],cin,sum[0],c[0]);
```

```
fulladder FA2(a[1],b[1],c[0],sum[1],c[1]);
```

```
fulladder FA3(a[2],b[2],c[1],sum[2],c[2]);
```

```
fulladder FA4(a[3],b[3],c[2],sum[3],c[3]);
```

```
fulladder FA5(a[4],b[4],c[3],sum[4],c[4]);
```

```
fulladder FA6(a[5],b[5],c[4],sum[5],c[5]);
```

```
fulladder FA7(a[6],b[6],c[5],sum[6],c[6]);
```

```
fulladder FA8(a[7],b[7],c[6],sum[7],carry);
```

```
endmodule
```

RESULT:

Date:

Ex No : 3 SIMULATION AND IMPLEMENTATION OF MULTIPLIER

AIM : To simulate and implement 4-bit multiplier in FPGA using Verilog code.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

THEORY:

4-BIT ARRAY MULTIPLIER

Binary multiplication is done by doing additions. Partial products are calculated by multiplying the multiplicand by each bit of the multiplier and then summing the partial products.

Example

								Inputs
				A3 B3	A2 B2	A1 B1	A0 B0	
				C	B0 x A3	B0 x A2	B0 x A1	B0 x A0
				+	B1 x A3	B1 x A2	B1 x A1	B1 x A0
				C	sum	sum	sum	sum
				+	B2 x A3	B2 x A2	B2 x A1	B2 x A0
				C	sum	sum	sum	sum
				+	B3 x A3	B3 x A2	B3 x A1	B3 x A0
				C	sum	sum	sum	sum
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Outputs

8. Type your source file and save.
9. Click on Simulation radio button on the left corner of the screen.
10. Select your filename in the hierarchy.
11. Expand ISIM Simulator and check for behaviour and simulate the program.
12. Select the necessary input values by right click on value -> Force Constant.
13. Enter the value in the Force to Value dialog box.
14. Click on "Run for specific time" in the toolbar.
15. Observe the waveforms.

IMPLEMENTATION

- Click on project, go to new source. Select implementation constraints file type and enter file name & click next, then finish
- Write the inputs, outputs & its pin location in proper format of .ucf file.
- Open main program & double click on synthesize-XST. After successful completion of synthesis, double click on implementation design. (Translate, Map, Place & Route)
- After Successful completion of implement design double click on Programming file Generation.
- After the successful completion of all these processes double click on configure Target Device and a new ISE iMPACT window open.
- Connect the Xilinx board to your PC using USB board.
- Double click on Boundary Scan. Check auto cable connection out.
- Click on File->Initialize chain.
- After that they ask for "Do You want to continue and assign configuration files?"
- Click on Yes and Select the generated bit-stream file.
- Click on open. After that they ask "Do you want to attach an SPI or BPI PROM to this device". Click on operation ->program. If programming successful then they show Program succeeded.
- Output can be viewed in the board by using LEDs and switches.

PROGRAM:

//Full Adder

```
module Full_Adder(input x, y, cin, output s, cout);  
wire c1,c2,c3;  
xor(s,x,y,cin);  
and(c1,x,y),(c2,x,cin),(c3,y,cin);  
or(cout,c1,c2,c3);  
endmodule
```

//4-Bit array multiplier

```
module Mul_4bit(input [3:0] Q,input [3:0] M,output [7:0]P);  
wire c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11;  
wire d1,d2,d3,d4,d5,d6,d7;  
wire e1,e2,e3;  
wire f1,f2,f3,f4,f5,f6,f7;  
wire g1,g2,g3,g4;  
and(c1,M[3],Q[1]),(c2,M[2],Q[2]),(c3,M[1],Q[3]),(c4,M[3],Q[0]),(c5,M[2],Q[1]),  
(c6,M[1],Q[2]),(c7,M[2],Q[0]),(c8,M[1],Q[1]),(c9,M[0],Q[2]),(c10,M[1],Q[0]),  
(c11,M[0],Q[1]),(P[0],M[0],Q[0]);  
Full_Adder fa1(c1,c2,c3,d2,d1);  
Full_Adder fa2(c4,c5,c6,d4,d3);  
Full_Adder fa3(c7,c8,c9,d6,d5);  
Full_Adder fa4(c10,c11,0,P[1],d7);  
and(e1,M[2],Q[3]),(e2,M[3],Q[2]),(e3,M[0],Q[3]);  
Full_Adder fa5(e1,e2,d1,f2,f1);  
Full_Adder fa6(d2,d3,f5,f4,f3);  
Full_Adder fa7(d4,e3,d5,f6,f5);  
Full_Adder fa8(d6,d7,0,P[2],f7);
```

```
and(g1,M[3],Q[3]);  
Full_Adder fa9(g1,f1,g2,P[6],P[7]);  
Full_Adder fa10(f2,f3,g3,P[5],g2);  
Full_Adder fa11(f4,0,g4,P[4],g3);  
Full_Adder fa12(f6,f7,0,P[3],g4);  
endmodule
```

Result:

Date:

Ex No :4 SIMULATION AND IMPLEMENTATION OF MUX/DEMUX

AIM : To simulate and implement 2X1 Multiplexer and Demultiplexer in FPGA using Verilog code in different modelings.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

THEORY:

A multiplexer is a data selector device that selects one input from several input lines, depending upon the enabled, select lines, and yields one single output.

A multiplexer of 2^n inputs has n select lines, are used to select which input line to send to the output.

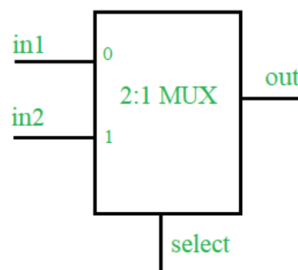


Fig: 2:1 Multiplexer

We'll code the 2:1 multiplexer in the following abstraction layers:

1. Structural modeling
2. Dataflow modeling
3. Behavioral modeling

1. Structural modeling:

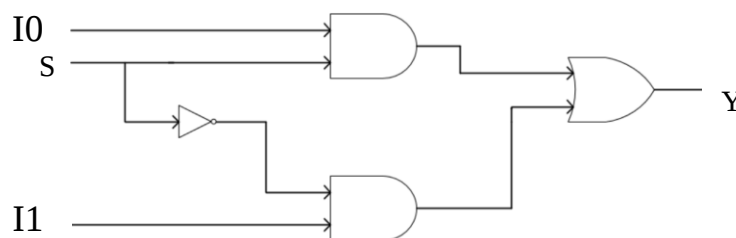


Fig: 2:1 MUX Verilog in structural modelling

In structural modeling, we describe the physical structure of a digital system. It is implemented using the logic gates in the circuit diagram. It gives us the internal hardware involved in the system. The gate-level modeling style uses the built-in basic logic gates predefined in Verilog.

2.Data flow modeling:

The dataflow modeling represents the flow of the data. It is described through the data flow through the combinational circuits rather than the logic gates used.

In Verilog, the assign statement is used in data-flow abstraction.

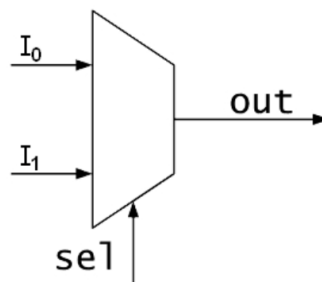


Fig. 2:1 MUX Verilog in Data Flow Model

3.Behavioral modeling:

The behavioral style, as the name suggests, describes the behavior of a circuit. It is the highest abstraction layer in the Verilog modeling of digital systems.

PROGRAM:

2:1 MUX in gate level model:

```
module mux_2x1_gl(input I0,I1,S,output Y);  
  wire sb,a,b;  
  not(sb,S);  
  and(a,sb,I0);  
  and(b,S,I1);  
  or(Y,a,b);  
endmodule
```

2:1 MUX Verilog in Data Flow Model:

```
module mux_2x1_df(in0, in1,sel,out);  
  input in0, in1, sel;  
  output out;  
  assign out=(sel)? in1:in0;  
endmodule
```

2:1 MUX in Behavioral Model:

```
module mux_2x1_b(in0, in1,sel,out);  
  input in0, in1, sel;  
  output reg out;  
  always @(*)  
  begin  
    if(sel)  
      out= in1;  
    else  
      out=in0;  
    end  
endmodule
```

RESULT:

Date:

Ex No :5 SIMULATION AND IMPLEMENTATION OF ENCODER AND DECODER

AIM :

To simulate and implement Encoder and Decoder in FPGA using Verilog code in different modelings.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

THEORY:

ENCODER:

In Digital System Circuit, an Encoder is a combinational circuit that takes 2^n input signal lines and encodes them into n output signal lines. When the enable is true i.e., the corresponding input signal lines display the equivalent binary bit output. For example, 8:3 Encoder has 8 input lines and 3 output lines, 4:2 Encoder has 4 input lines and 2 output lines, and so on.

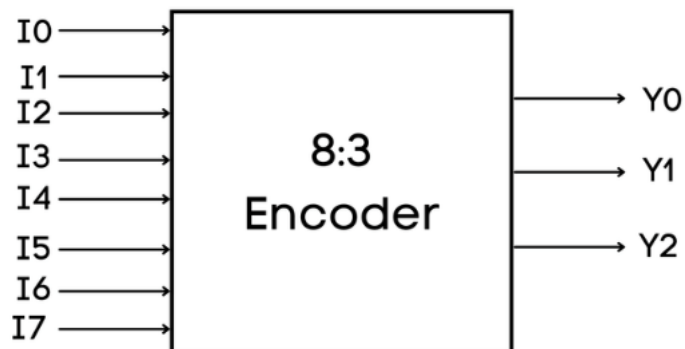
An 8:3 Priority Encoder has seven input lines i.e., i0 to i7, and three output lines y2, y1, and y0. In 8:3 Priority Encoder i7 have the highest priority and i0 the lowest.

Truth Table:

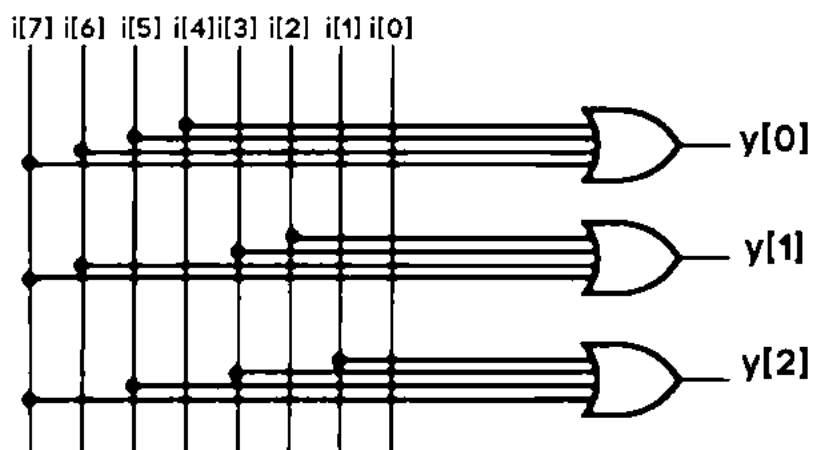
	Input								Output		
en	i7	i6	i5	i4	i3	i2	i1	i0	y2	y1	y0
0	x	x	x	x	x	x	x	x	z	z	z
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	x	0	0	1
1	0	0	0	0	0	1	x	x	0	1	0

	Input								Output		
en	i7	i6	i5	i4	i3	i2	i1	i0	y2	y1	y0
1	0	0	0	0	1	x	x	x	0	1	1
1	0	0	0	1	x	x	x	x	1	0	0
1	0	0	1	x	x	x	x	x	1	0	1
1	0	1	x	x	x	x	x	x	1	1	0
1	1	x	x	x	x	x	x	x	1	1	1

Logic Symbol:



Logic diagram:



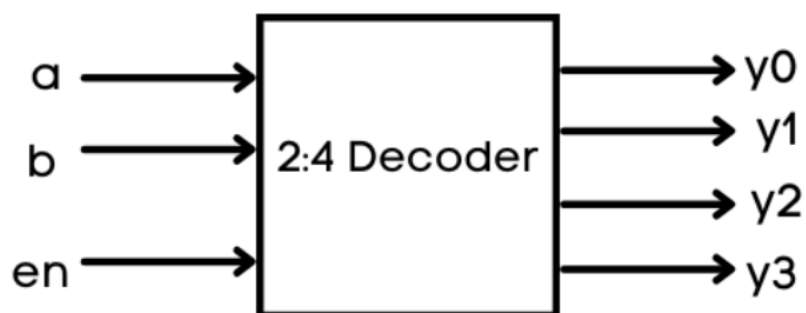
2:4 DECODER:

A decoder is a combinational logic circuit that has 'n' input signal lines and 2^n output lines. In the 2:4 decoder, we have 2 input lines and 4 output lines. In addition, we provide 'enable' to the input to ensure the decoder is functioning whenever enable is 1 and it is turned off when enable is 0. The truth table, logic diagram, and logic symbol are given below:

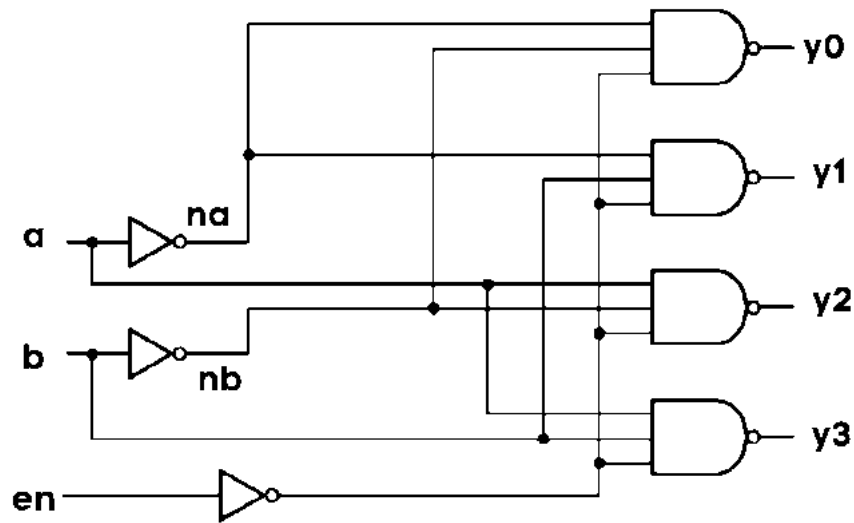
Truth Table:

En	Input		Output			
	a	b	y3	y2	y1	y0
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Logic Symbol:



Logic Diagram:



PROGRAM:

Encoder:

1. Gate Level Modelling:

```
module priorityencoder83_gate(en,i,y);
input en;
input [7:0]i;
output [2:0]y;
wire temp1,temp2,temp3;
or o1(temp1,i[4],i[5],i[6],i[7]);
or o2(temp2,i[2],i[3],i[6],i[7]);
or o3(temp3,i[1],i[3],i[5],i[7]);
and a1(y[2],temp1,en);
and a2(y[1],temp2,en);
and a3(y[0],temp3,en);
endmodule
```

2. Data Flow Modeling:

```
module priorityencoder83_dataflow(en,i,y);
input en;
input [7:0]i;
output [2:0]y;
assign y[2]=i[4] | i[5] | i[6] | i[7] &en;
assign y[1]=i[2] | i[3] | i[6] | i[7] &en;
assign y[0]=i[1] | i[3] | i[5] | i[7] &en;
endmodule
```

3. Behavior Modeling:

```
module priorityencoder_83(en,i,y);
input en;
input [7:0]i;
output reg [2:0]y;
always @(en,i)
begin
    if(en==1)
        begin
            if(i[7]==1) y=3'b111;
            else if(i[6]==1) y=3'b110;
            else if(i[5]==1) y=3'b101;
            else if(i[4]==1) y=3'b100;
            else if(i[3]==1) y=3'b011;
            else if(i[2]==1) y=3'b010;
            else if(i[1]==1) y=3'b001;
            else
                y=3'b000;
        end
        else y=3'bzzz;
    end
endmodule
```


Decoder:

1. Gate Level Modeling:

```
module decoder24_gate(en,a,b,y);
input en,a,b;
output [3:0]y;

wire enb,na,nb;
not n0(enb,en);
not n1(na,a);
not n2(nb,b);
nand n3(y[0],enb,na,nb);
nand n4(y[1],enb,na,b);
nand n5(y[2],enb,a,nb);
nand n6(y[3],enb,a,b);
endmodule
```

2. Data Flow Modeling:

```
module decoder24_df(en,a,b,y);
input en,a,b;
output [3:0]y;
wire enb,na,nb;
assign enb = ~en;
assign na = ~a;
assign nb = ~b;
assign y[0] = ~(enb & na & nb);
assign y[1] = ~(enb & na & b);
assign y[2] = ~(enb & a & nb);
assign y[3] = ~(enb & a & b);
endmodule
```

3. Behavioral Modeling:

```
module decoder24_b(en,a,b,y);
input en,a,b;
output reg [3:0]y;
always @(en,a,b)
begin
    if(en==0)
```

```
begin
  if(a==1'b0 & b==1'b0) y=4'b1110;
  else if(a==1'b0 & b==1'b1) y=4'b1101;
  else if(a==1'b1 & b==1'b0) y=4'b1011;
  else if(a==1 & b==1) y=4'b0111;
  else y=4'bxxxx;
end
else
  y=4'b1111;
end
endmodule
```

RESULT:

Date:

Ex No : 6 SIMULATION AND IMPLEMENTATION OF BASIC SEQUENTIAL CIRCUITS

AIM :

To simulate and implement basic sequential circuits in FPGA using Verilog code.

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

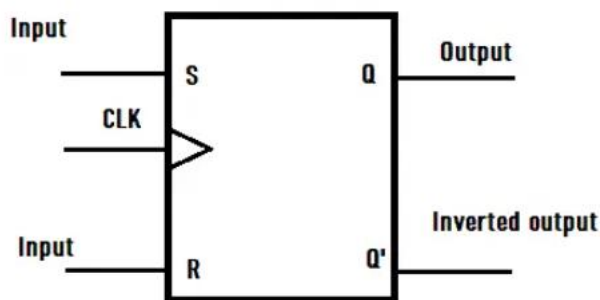
THEORY:

Flip Flops are the basic building blocks of sequential circuits. They are memory elements made by connecting logic gates. They can shift between two states (0 and 1) and hence, formally called bi-stable multivibrator. Using flip flops, we build complex circuits such as RAMs, Shift Registers, etc.

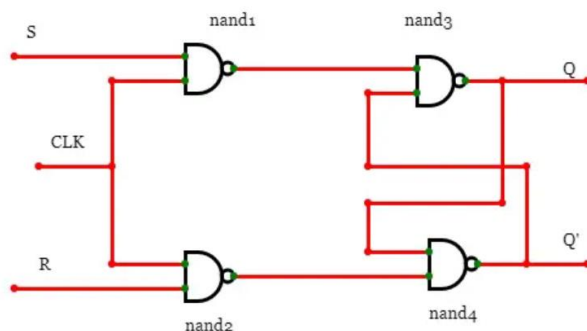
SR FLIPFLOP:

An SR Flip Flop is short for Set-Reset Flip Flop. It has two inputs S(Set) and R(Reset) and two outputs Q(normal output) and Q'(inverted output).

Symbol:



Logic Circuit:



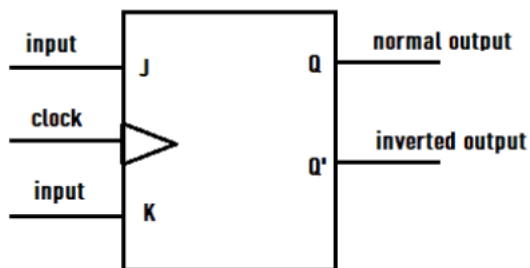
Truth Table:

S	R	Q	Q'	State
0	0	Q	Q'	No change
0	1	0	1	Reset State
1	0	1	0	Set State
1	1	X	X	Invalid

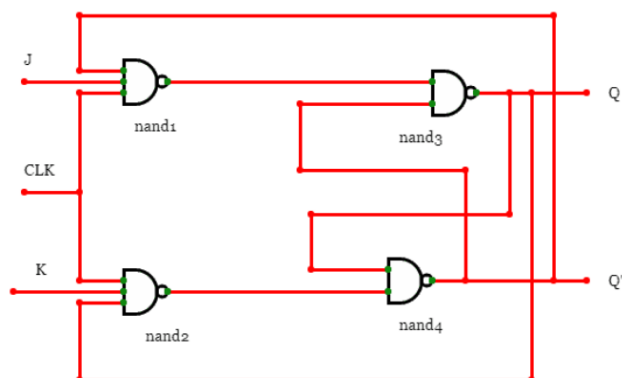
JK FLIPFLOP:

The J-K flip-flop is the most versatile of the basic flip flops. The JK flip flop is a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic 1. Due to this additional clocked input, a JK flip-flop has four possible input combinations, “logic 1”, “logic 0”, “no change” and “toggle”.

Symbol:



Logic Circuit:



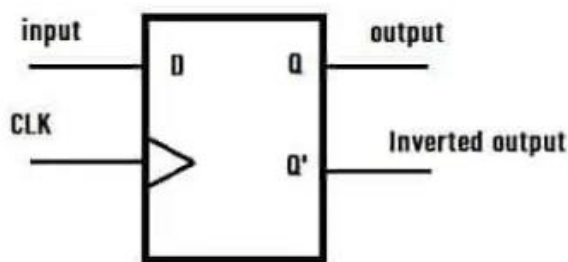
Truth Table:

J	K	Q	Q'	State
0	0	Q	Q'	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	$(Q_{\text{prev}})'$	Q_{prev}	Toggle

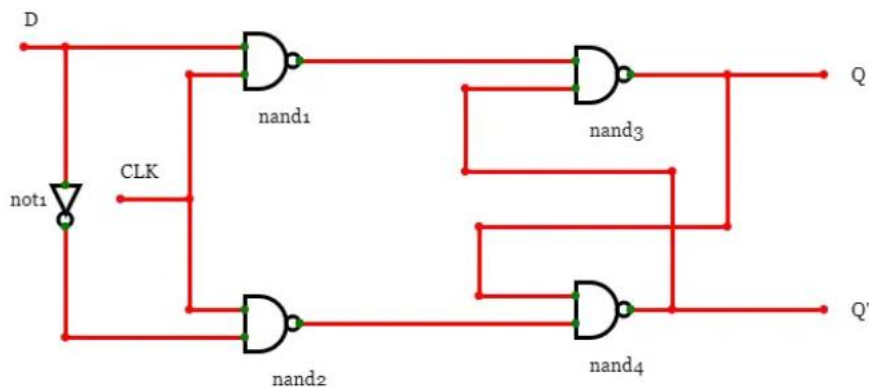
D FLIPFLOP:

A D flip-flop stands for data or delay flip-flop. The outputs of this flip-flop are equal to the inputs.

Symbol:



Logic Circuit:



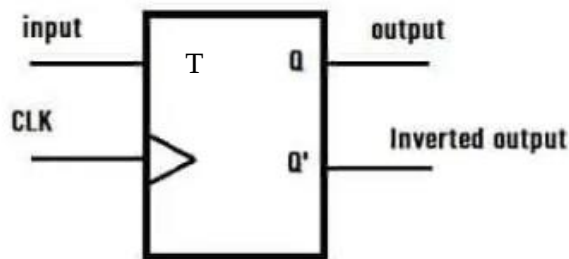
Truth Table:

D	Q	Q'	State
0	0	1	Reset State
1	1	0	Set State

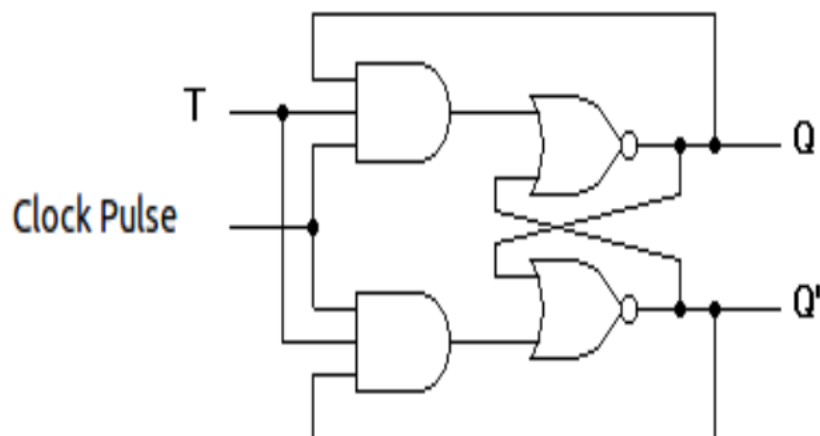
T Flip-Flop

The toggle, or T, flip-flop is a two-input flip-flop. The inputs are the toggle (T) input and a clock (CLK) input. If the toggle input is HIGH, the T flip-flop changes state (toggles) when the clock signal is applied. If the toggle input is LOW, the T flip-flop holds the previous state.

Symbol:



Logic Circuit:



Truth Table:

T	Q	Q _{next}	State
0	Q	Q	Hold State
1	Q	Q _{bar}	Toggle

PROGRAM:**SR FLIPFLOP:**

```
module srff_behave(s,r,clk, q, qbar);
input s,r,clk;
output reg q, qbar;
always@(posedge clk)
begin
    if(s == 1)
    begin
        q = 1;
        qbar = 0;
    end
    else if(r == 1)
    begin
        q = 0;
        qbar =1;
    end
    else if(s == 0 & r == 0)
    begin
        q <= q;
        qbar <= qbar
    end
end
endmodule
```

JK FLIPFLOP:

```
module jkff_gate(q,qbar,clk,j,k);
input j,k,clk;
output q,qbar;
wire nand1_out;
wire nand2_out;
nand(nand1_out, j,clk,qbar);
nand(nand2_out, k, clk,q);
nand(q, qbar, nand1_out);
nand(qbar, q, nand2_out);
endmodule
```

D FLIPFLOP:

```
module dff_behavioral(d,clk,clear,q,qbar);
input d, clk, clear;
output reg q, qbar;
always@(posedge clk or posedge clear)
begin
```

```

        if(clear== 1)
            q <= 0;
            qbar <= 1;
        else
            q <= d;
            qbar = !d;
        end
    endmodule

```

T FLIPFLOP:

```

module tff(clk,t,q);
input clk,t;
output reg q;
always @ (posedge clk)begin
    if(t == 0)
        q <= q;
    else
        q = ~q;
    end
endmodule

```

RESULT:

Ex No :7 SIMULATION AND IMPLEMENTATION OF UNIVERSAL SHIFT REGISTER

SOFTWARE USED:

- Xilinx 8.1 ISE Sim
- Spartan 3E

The Shift Register is a sequential logic circuit that can be used for the storage or the transfer of binary data. Universal shift register is capable of converting input data to parallel or serial which also does shifting of data bidirectional, unidirectional (SISO, SIPO, PISO, PIPO) and are parallel load this is called as Universal shift register.

4-bit universal shift register.

Fig. Block Diagram of 4-bit USR

Table: Function Table

Mode Control		Register operation
S1	S0	
0	0	No change
0	1	Shift left
1	0	Shift right
1	1	Parallel load

PROGRAM:

```

module SHFTREG(I,select,lfin,rtin,A,CLK,Clr);
input[3:0]I;    //Parallel input
input[1:0]select;    //Mode select
input lfin,rtin,CLK,Clr; //Serial inputs,clock,clear
output[3:0]A;    //Parallel output

//Instantiate the four stages
stage ST0(A[0],A[1],lfin,I[0],A[0],select,CLK,Clr);
stage ST1(A[1],A[2],A[0],I[1],A[1],select,CLK,Clr);
stage ST2(A[2],A[3],A[1],I[2],A[2],select,CLK,Clr);
stage ST3(A[3],rtin,A[2],I[3],A[3],select,CLK,Clr);
endmodule

//One stage of shift register
module stage(i0,i1,i2,i3,Q,select,CLK,Clr);
input i0,i1,i2,i3,CLK,Clr;
input[1:0]select;
output Q;
reg Q;
reg D;
//4x1 multiplexer
always@(i0 or i1 or i2 or i3 or select)
    case(select)

```

```
2'b00:D=i0;
2'b01:D=i1;
2'b10:D=i2;
2'b11:D=i3;
endcase
//D flip-flop
always@(posedge CLK or negedge Clr)
  if(~Clr) Q=1'b0;
  else Q=D;
endmodule
```

RESULT:

Date:

EX. NO. 8 LAYOUT GENERATION FOR UNIVERSAL LOGIC GATES

AIM:

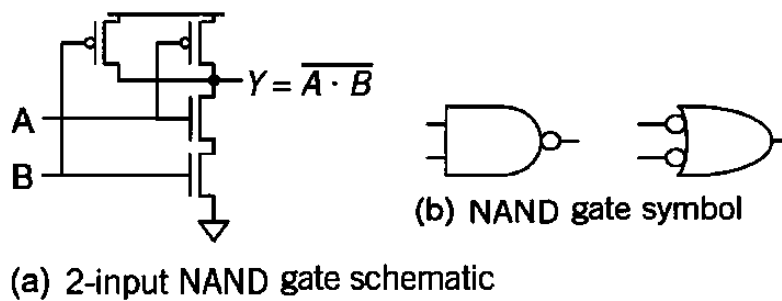
To design and simulate NAND and NOR gates using Microwind by generating their layouts.

TOOLS REQUIRED:

1. MICROWIND V3.1

THEORY:

NAND GATE:

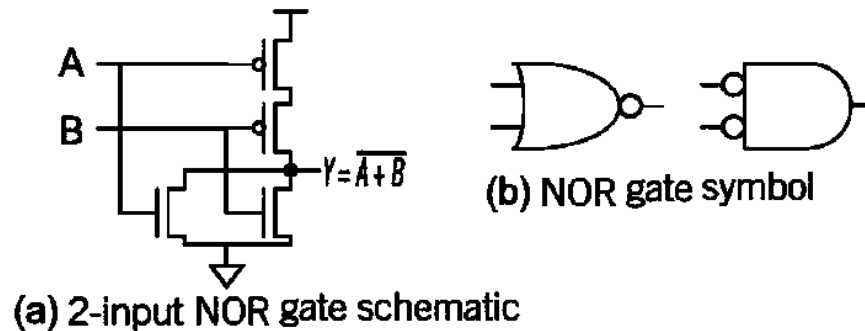


The figure shows a 2-input Complementary MOS NAND gate. It consists of two series NMOS transistors between Y and Ground and two parallel PMOS transistors between Y and VDD. If either input A or B is logic 0, at least one of the NMOS transistors will be OFF, breaking the path from Y to Ground. But at least one of the pMOS transistors will be ON, creating a path from Y to VDD. Hence, the output Y will be high. If both inputs are high, both of the nMOS transistors will be ON and both of the pMOS transistors will be OFF. Hence, the output will be logic low.

Truth Table

A	B	Pull-down Network	Pull-up Network	Output Y
0	0	OFF	ON	1
0	1	OFF	ON	1
1	0	OFF	ON	1
1	1	ON	OFF	0

NOR GATE:



A 2-input NOR gate is shown in the figure. The NMOS transistors are in parallel to pull the output low when either input is high. The PMOS transistors are in series to pull the output high when both inputs are low, as given in below table. The output is never left floating.

Truth Table

A	B	Pull-down Network	Pull-up Network	Output Y
0	0	OFF	ON	1
0	1	ON	OFF	0
1	0	ON	OFF	0
1	1	ON	OFF	0

PROCEDURE:

1. Double click on Microwind and go to file and click new.
2. Select MOS Generator and select PMOS device, type 2 in 'Nbr of Fingers' and click Generate Device.
3. Similarly generate NMOS device and place it.
4. Select Polysilicon from the Palette and connect the gates of PMOS and NMOS
5. Now connect the two transistors using a metal1 layer from the Palette.
6. Now make connections for inputs and output using corresponding layers.
7. Choose Vdd(supply) from Palette and place it in the metal 1 and N-well of PMOS.
8. Choose Vss(ground) from the Palette and place it in the metal 1 of nNMOS.
9. Now provide inputs, by choosing 'Add a Clock' from the Palette, select assign and place it at the input side.
10. Now output is placed by choosing 'Visible node symbol' from the Palette and click assign to place it at the output port.
11. Save the file and select 'Run Simulation' from the Menu bar.
12. Output waveform is generated in a new window.

RESULT:

Date:

EX. NO. 9 LAYOUT GENERATION FOR FLIPFLOPS

AIM:

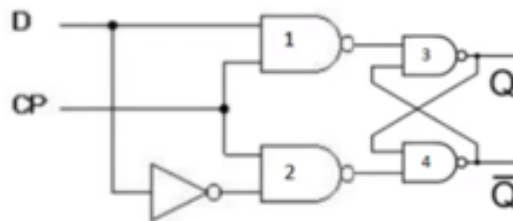
To design and simulate flipflops using Microwind and DSCH V3.1 by generating their layouts.

TOOLS REQUIRED:

1. MICROWIND V3.1
2. DSCH V3.1

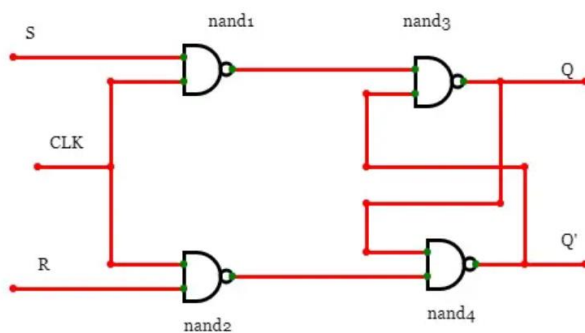
THEORY:

D-FLIPFLOP:



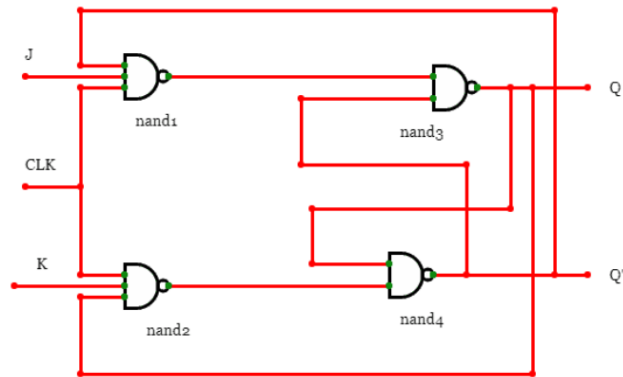
D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal. The next state equation is $Q(t+1) = D$.

SR FLIPFLOP:



An SR Flip Flop is short for Set-Reset Flip Flop. It has two inputs S(Set) and R(Reset) and two outputs Q(normal output) and Q'(inverted output).

JK FLIPFLOP:



The J-K flip-flop is the most versatile of the basic flip flops. The JK flip flop is a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic 1. Due to this additional clocked input, a JK flip-flop has four possible input combinations, “logic 1”, “logic 0”, “no change” and “toggle”.

PROCEDURE:

1. Open the DSCH2
2. Drag the components like nand gate, voltage source, ground, switch and LED from the symbol library.
3. Connect the circuit as in the circuit diagram.
4. Save the circuit & run the simulation. Make verilog file go to Microwind and compile the verilog file saved in DSCH2
5. Compile it and obtain the layout diagram & simulate to get the waveform.

Result:

Date:

EX. NO. 10 LAYOUT GENERATION FOR A 4-BIT SYNCHRONOUS COUNTER

AIM:

To design and simulate a 4-bit synchronous counter using Microwind and DSCH V3.1 by generating its layout.

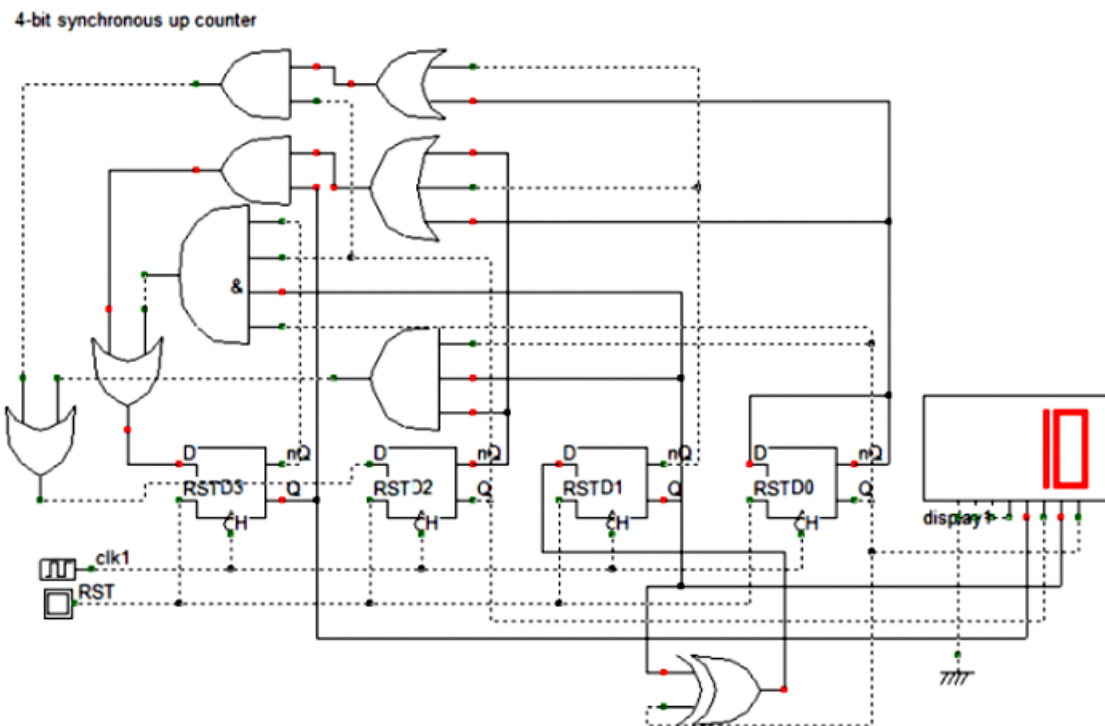
TOOLS REQUIRED:

3. MICROWIND V3.1
4. DSCH V3.1

THEORY:

4-BIT SYNCHRONOUS COUNTER:

Synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.



A 4-bit synchronous up counter, we will need four flip-flops. These flip-flops will have the same RST signal and the same CLK signal. Here D flip-flops are used to design this counter. The 4-bit synchronous up counter should follow the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0.

PROCEDURE:

1. Open the DSCH2
2. Drag the components like pmos, nmos, voltage source, ground, and LED from the symbol library.
3. Connect the circuit as in the circuit diagram.
4. Save the circuit & run the simulation Make verilog file go to Microwind and compile the verilog file saved in DSCH2
5. Compile it and obtain the layout diagram & simulate to get the waveform.

Result: